



# Experimental Comparison of Crypto-processors Architectures for Elliptic and Hyper-Elliptic Curves Cryptography

Gabriel Gallin, Arnaud Tisserand, Nicolas Veyrat-Charvillon

## ► To cite this version:

Gabriel Gallin, Arnaud Tisserand, Nicolas Veyrat-Charvillon. Experimental Comparison of Crypto-processors Architectures for Elliptic and Hyper-Elliptic Curves Cryptography. CryptArchi: 13th International Workshops on Cryptographic Architectures Embedded in Reconfigurable Devices, Jun 2015, Leuven, Belgium. . hal-01197048

**HAL Id: hal-01197048**

**<https://inria.hal.science/hal-01197048>**

Submitted on 11 Sep 2015

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Experimental Comparison of Crypto-processors Architectures for Elliptic and Hyper-Elliptic Curves Cryptography

**Gabriel GALLIN**, Arnaud TISSERAND and  
Nicolas VEYRAT-CHARVILLON

CNRS – IRISA – University Rennes 1 – CAIRN  
HAH Project

13th CryptArchi Workshop: June 29-30th, 2015



IRISA

UNIVERSITÉ DE  
RENNES 1

CominLabs



Henri Lebesgue  
CENTRE DE RECHERCHES



informatics mathematics  
*inria*

ENSSAT  
LANNION



# Summary

- 1 Context & Motivations
- 2 Proposed Crypto-processor(s)
- 3 Experiments & Comparisons
- 4 Conclusion & Perspectives

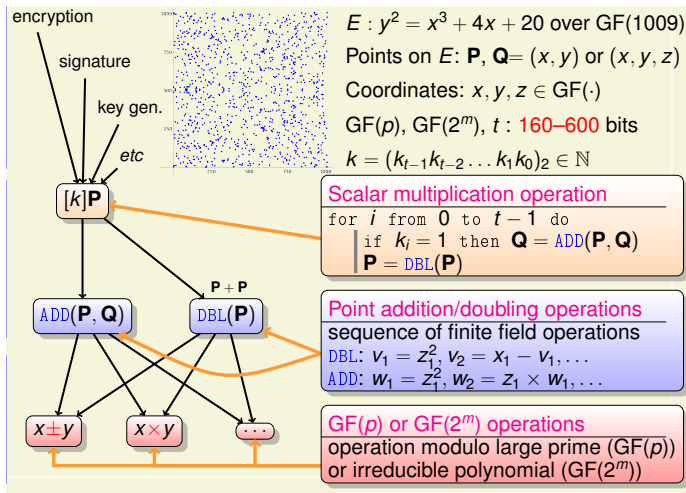
# Summary

- 1 Context & Motivations
- 2 Proposed Crypto-processor(s)
- 3 Experiments & Comparisons
- 4 Conclusion & Perspectives

# Asymmetric Cryptography: ECC and HECC

- Cryptographic primitives for protocols such as **digital signature**, **exchange of secret keys** and some **specific encryption schemes**
- Elliptic Curve Cryptography (ECC):
  - Actual standard for public key crypto-systems
  - **Better performance** and **lower cost** than RSA
- Hyper-Elliptic Curve Cryptography (HECC):
  - Evolution of ECC focusing a larger set of curves
  - Studied for future generations of asymmetric crypto-systems

# Operations for (H)ECC



Metric for algorithms efficiency: number of multiplications (M) and squares (S) in  $\mathbb{F}_p$

# ECC versus HECC

	size of $\mathbb{F}_p$ & scalar	ADD	DBL	source
ECC	$\ell_{\text{ECC}}$	12M + 2S	7M + 3S	[1]
HECC	$\ell_{\text{HECC}} \approx \frac{1}{2}\ell_{\text{ECC}}$	40M + 4S	38M + 6S	[3]

- ECC:
  - Size of  $\mathbb{F}_p$  and scalar  $2\times$  larger
  - Simpler ADD and DBL operations
- HECC:
  - Smaller  $\mathbb{F}_p$  and scalar
  - More operations in  $\mathbb{F}_p$  for ADD and DBL
- In theory, HECC naturally better than ECC

# Objectives

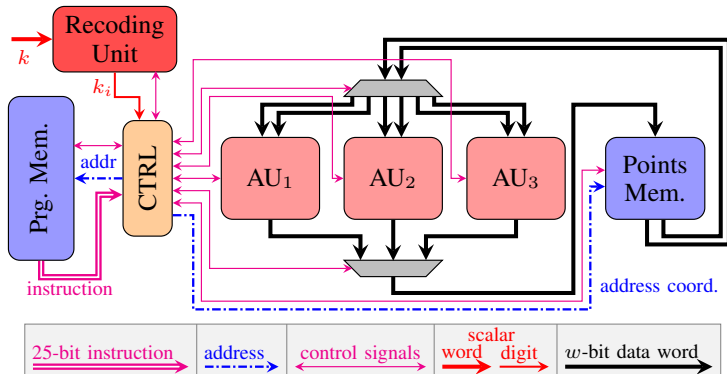
- Design hardware crypto-processors based on **customisable** architecture
- Implementation of small circuits on **FPGA** and ASIC
- Study various trade-offs between:
  - **Computation speed**
  - **Area cost**
  - Energy consumption and protection against physical attacks
- **Experimental comparisons** of different architecture configurations  
→ choice of the architecture parameters for the crypto-processor



# Summary

- 1 Context & Motivations
- 2 Proposed Crypto-processor(s)**
- 3 Experiments & Comparisons
- 4 Conclusion & Perspectives

# Developed Processor



- Customizable number of arithmetic units over  $\mathbb{F}_p$ :  $\pm, \times, \div$   
 $\rightarrow n_M$  multipliers of size  $n_B$
- $w$  : size of data words

# Architecture Parameters

- $w = 32$  bits, for small processors
- 1 adder/subtractor ( $\pm$ ), small and fast
- 1 inverter ( $\div$ ), sufficient for the computations
- $n_M$  multipliers ( $\times$ ):
  - Based on Montgomery algorithm for modular multiplication [5]
  - $n_B$  : number of parallel active words in the multiplier
  - 3-stage pipeline
- Classical key recoding techniques from literature:
  - standard binary, window  $\lambda$ NAF methods with  $\lambda \in \{2...4\}$

# Summary

- 1 Context & Motivations
- 2 Proposed Crypto-processor(s)
- 3 Experiments & Comparisons**
- 4 Conclusion & Perspectives

# Details of the Experiments

- **Objective:** compare versions of the processor for various ( $n_M$ ,  $n_B$ )
- Implementation on Xilinx Spartan 6 LX75 FPGA
- **No DSP blocks** (for ASIC compatibility)
- Design tools:
  - VHDL and assembly **code generation**: Python scripts
  - Design implementation: Xilinx design environment ISE 14.6
  - Simulations of complete scalar multiplications with Modelsim  
Theoretical validation with SAGE (with more than 10k vectors)
- Implementation: **translate, map, place and route** of full processor
- **Same optimisation efforts** for ECC and HECC

# Impact of Key Recoding Unit

- Various recoding techniques proposed to **reduce the number of curve operations**:
  - BIN: standard binary from left to right
  - NAF: non-adjacent form
  - $\lambda$ NAF: window methods with  $\lambda \in \{3, 4\}$
- Implementation results for an ECC processor ( $n_M = 1$ ,  $n_B = 1$ )

recoding	BIN	NAF	3NAF	4NAF
area, slices (FF/LUT)	517 (1347/1433)	536 (1366/1445)	560 (1370/1454)	547 (1374/1460)
frequency, MHz	229	234	235	231

# Results for ECC (256 bits) and HECC (128 bits) (1/2)

	$n_M$	BRAM	$n_B = 1$		$n_B = 2$		$n_B = 4$	
			area slices (FF/LUT)	freq. MHz	area slices (FF/LUT)	freq. MHz	area slices (FF/LUT)	freq. MHz
ECC	1	3	547 (1374/1460)	231	573 (1476/1625)	233	673 (1674/1875)	233
	2	3	722 (1776/1903)	220	811 (1979/2210)	227	942 (2377/2701)	220
	3	3	810 (2174/2236)	221	915 (2480/2698)	215	1130 (3077/3430)	214
	4	3	952 (2569/2656)	215	1100 (2977/3282)	217	1512 (3771/4293)	216
	5	3	1064 (2982/3136)	210	1405 (3492/3902)	206	1722 (4487/5122)	209
HECC	1	4	514 (1336/1374)	235	549 (1434/1513)	234		
	2	4	646 (1716/1783)	220	737 (1912/2055)	234		
	3	4	732 (2092/2075)	224	826 (2386/2485)	225		
	4	4	870 (2476/2424)	218	1022 (2868/2987)	214		
	5	4	976 (2865/2773)	219	1115 (3355/3465)	210		
	6	4	1089 (3233/3092)	203	1240 (3821/3908)	208		
	7	4	1145 (3601/3426)	213	1372 (4287/4365)	205		
	8	4	1281 (3981/3809)	191	1552 (4765/4890)	183		
	9	4	1379 (4363/4051)	202	1691 (5245/5277)	199		
	10	4	1543 (4739/4435)	196	1856 (5719/5801)	198		
	11	4	1547 (5114/4750)	189	1936 (6192/6240)	198		
	12	4	1738 (5499/5128)	191	2100 (6675/6771)	188		

# Results for ECC (256 bits) and HECC (128 bits) (2/2)

- Average computation time (ms) for 50  $[k]P$ :

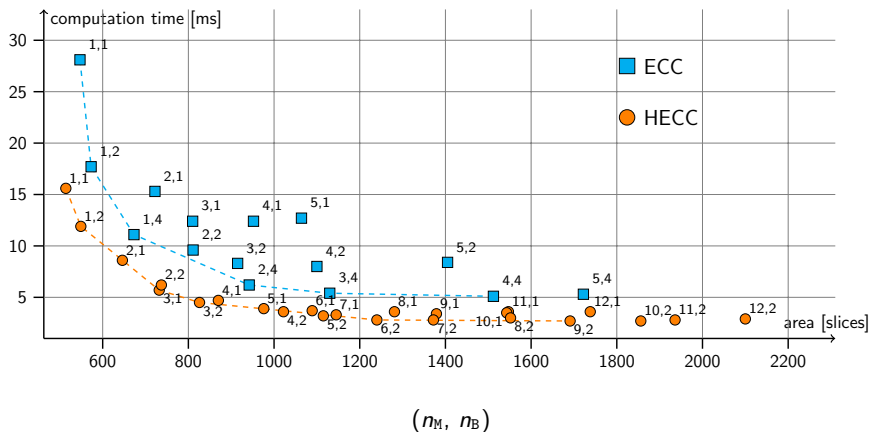
	$n_B$	$n_M$											
		1	2	3	4	5	6	7	8	9	10	11	12
HECC	1	15.6	8.6	5.7	4.7	3.9	3.7	3.3	3.6	3.4	3.5	3.6	3.6
	2	11.9	6.2	4.5	3.6	3.2	2.8	2.8	3.0	2.7	2.7	2.8	2.9
ECC	1	28.1	15.3	12.4	12.4	12.7							
	2	17.7	9.6	8.3	8.0	8.4							
	4	11.1	6.2	5.4	5.1	5.3							

- Standard deviation for 1000  $[k]P$ :

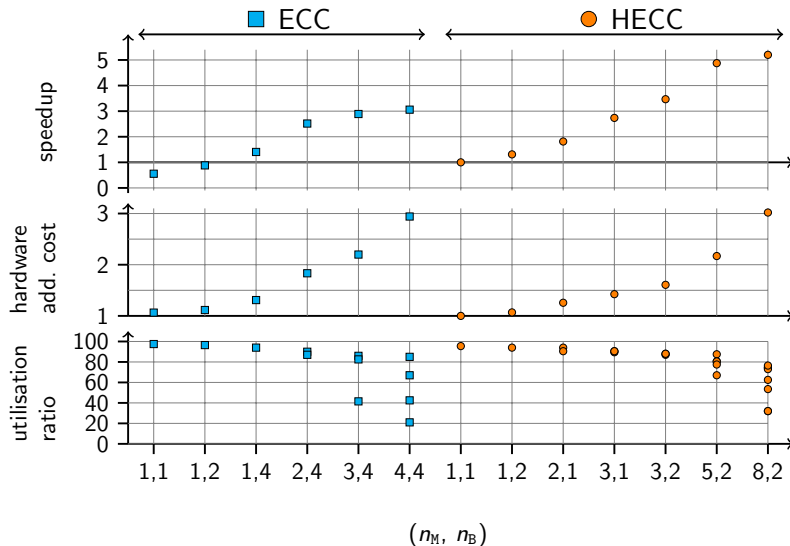
configuration	ECC (1,1)	ECC (3,4)	HECC (1,1)	HECC (6,2)
average time [ms]	28.2	5.3	15.5	2.8
std. deviation [ms]	0.289	0.056	0.324	0.045



# Comparison of Various Configurations for our Processor



# Relative Efficiency



# Comparison with Literature

Source	FPGA	area slices / DSP	freq. MHz	[k]P duration ms
ECC 1,2	Spartan 6	573 / 0	233	17.7
ECC 1,4		673 / 0	233	11.1
ECC 2,4		942 / 0	220	6.2
ECC 3,4		1 130 / 0	214	5.4
[4]	Virtex-5	1 725 / 37	291	0.38
	Virtex-4	4 655 / 37	250	0.44
[2]	Virtex-4	13 661 / 0	43	9.2
		20 123 / 0	43	7.7

# Summary

- 1 Context & Motivations
- 2 Proposed Crypto-processor(s)
- 3 Experiments & Comparisons
- 4 Conclusion & Perspectives

# Conclusion

- Implementation of ECC/HECC processors  $\Rightarrow$  cost/performance trade-offs
- We are able to select the appropriate number of multipliers and their size
- Experimental results: HECC always has **better performance** compared to ECC

# Perspectives

- Improve arithmetic units (especially add DSP blocks)
- Study resistance to physical attacks (SCA<sup>1</sup>, faults injection) of processors with good time/area trade-offs

---

<sup>1</sup>Side Channel Attacks

This work is partly funded by

**PAVOIS** project (ANR-12-BS02-002-01) <http://pavois.irisa.fr/>

**HAH** project <http://h-a-h.inria.fr/>

---

# Thank you for your attention

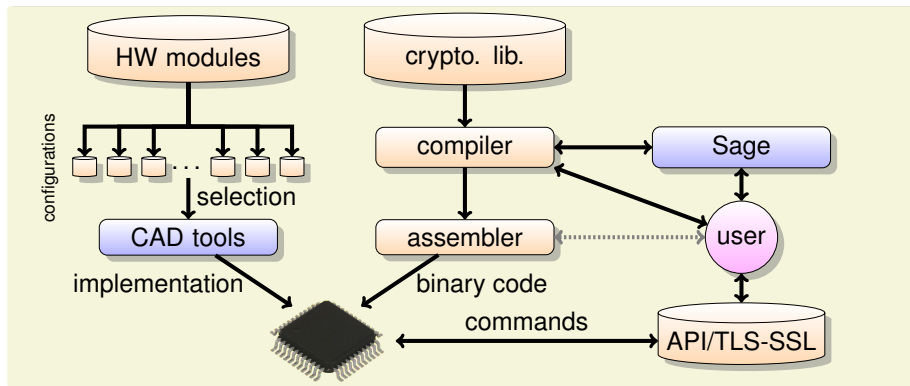


# References

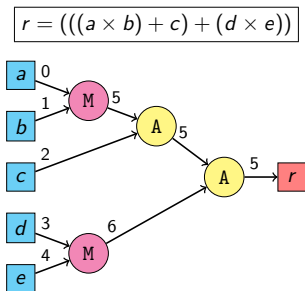
- [1] D. J. Bernstein and T. Lange.  
Explicit-formulas database.  
<http://hyperelliptic.org/EFD/>.
- [2] S. Ghosh, M. Alam, D. Roychowdhury, and I.S. Gupta.  
Parallel crypto-devices for  $\text{GF}(p)$  elliptic curve multiplication resistant against side channel attacks.  
*Computers and Electrical Engineering*, 35(2):329–338, March 2009.
- [3] T. Lange.  
Formulae for Arithmetic on Genus 2 Hyperelliptic Curves.  
*Applicable Algebra in Engineering, Communication and Computing*, 15(5):295–328, February 2005.
- [4] Y. Ma, Z. Liu, W. Pan, and J. Jing.  
A high-speed elliptic curve cryptographic processor for generic curves over  $\text{GF}(p)$ .  
In *Proc. 20th International Workshop on Selected Areas in Cryptography (SAC)*, volume 8282 of *LNCS*, pages 421–437.  
Springer, August 2013.
- [5] P. L. Montgomery.  
Modular multiplication without trial division.  
*Mathematics of Computation*, 44(170):519–521, April 1985.



# Implementation Flow



# Assembly Language: a Basic Example



1	read <code>fu_mul_0</code> , 0, 1	read operands <i>a</i> and <i>b</i>
2	launch <code>fu_mul_0</code>	launch <i>ab</i>
3	read <code>fu_mul_1</code> , 3, 4	read operands <i>d</i> and <i>e</i>
4	launch <code>fu_mul_1</code>	launch <i>de</i>
5	wait <code>fu_mul_0</code>	wait until the end of <i>ab</i>
6	write <code>fu_mul_0</code> , 5	write the result of <i>ab</i>
7	set OPMODE, 0	set addition (+) mode
8	read <code>fu_add_sub_0</code> , 5, 2	read <i>ab</i> and operand <i>c</i>
9	launch <code>fu_add_sub_0</code>	launch ( <i>ab</i> ) + <i>c</i>
10	wait <code>fu_mul_1</code>	wait until the end of <i>de</i>
11	write <code>fu_mul_1</code> , 6	write the result of <i>de</i>
12	wait <code>fu_add_sub_0</code>	wait until the end of ( <i>ab</i> ) + <i>c</i>
13	write <code>fu_add_sub_0</code> , 5	write the result of ( <i>ab</i> ) + <i>c</i>
14	read <code>fu_add_sub_0</code> , 5, 6	read ( <i>ab</i> ) + <i>c</i> and <i>de</i>
15	launch <code>fu_add_sub_0</code>	launch (( <i>ab</i> ) + <i>c</i> ) + ( <i>de</i> )
16	wait <code>fu_add_sub_0</code>	wait until the end of (( <i>ab</i> ) + <i>c</i> ) + ( <i>de</i> )
17	write <code>fu_add_sub_0</code> , 5	write (( <i>ab</i> ) + <i>c</i> ) + ( <i>de</i> )

# FPGA characteristics

FPGA	Spartan 6	Virtex-4 LX200 [2]	Virtex-5 LX110T [4]	Virtex-4 LX100 [4]
number of slices	11662	89088	17280	49152
number of FF	93296	178176	69120	98304
number of LUT	46648	178176	69120	98304